# Automating the processing of Earth observation data

**Keith Golden** **Wanlin Pang*** **Ramakrishna Nemani** **Petr Votava**
NASA Ames Research Center
MS 269-2
Moffett Field, CA 94035
{kgolden | wpang}@email.arc.nasa.gov

University of Montana
School of Forestry
Missoula, MT 59812
{nemani | votava}@ntsg.umt.edu

## 1 Introduction

NASA's vision for Earth science is to build a "sensor web": an adaptive array of heterogeneous satellites and other sensors that will track important events, such as storms, and provide real-time information about the state of the Earth to a wide variety of customers. Achieving this vision will require automation not only in the scheduling of the observations but also in the processing of the resulting data. To address this need, we are developing a planner-based agent to automatically generate and execute data-flow programs to produce the requested data products.

### 1.1 TOPS Case Study

As a demonstration of our approach, we are applying our agent, called IMAGEbot, to the Terrestrial Observation and Prediction System (TOPS, http://www.forestry.umt.edu/ntsg/Projects/TOPS/), an ecological forecasting system that assimilates data from Earth-orbiting satellites and ground weather stations to model and forecast conditions on the surface, such as soil moisture, vegetation growth and plant stress (Nemani *et al.* 2002). Prospective customers of TOPS include scientists, farmers and fire fighters. With such a variety of customers and data sources, there is a strong need for a flexible mechanism for producing the desired data products for the customers, taking into account the information needs of the customer, data availability, deadlines, resource usage (some scientific models take many hours to execute) and constraints based on context (a scientist with a palmtop computer in the field has different display requirements than when sitting at a desk). IMAGEbot provides such a mechanism, accepting goals in the form of descriptions of the desired data products.

The goal of the TOPS system is the monitoring and prediction of changes in key environmental variables. Early warnings of potential changes in these variables, such as soil moisture, snow pack, primary production and stream flow, could enhance our ability to make better socio-economic decisions relating to natural resource management and food production(Nemani *et al.* 2000). The accuracy of such warnings depends on how well the past, present and future conditions of the ecosystem are characterized.

*QSS Group Inc



The overall data flow through the system is depicted in Figure 1. The inputs needed by TOPS include:

- Fractional Photosynthetically Active Radiation (FPAR) and Leaf Area Index (LAI)
- Temperatures (minimum, maximum and daylight average)
- Precipitation
- Solar Radiation
- Humidity

We have several potential candidate data sources at the beginning of each model run. The basic properties of the inputs are listed in Table 1. Even with this fairly small model, there is a good variety of inputs we need to select from, depending on our goal.

In addition to the attributes listed in the table, data sources also vary in terms of quality and availability — some inputs are not always available even though they should be. For example, both the Terra and Aqua satellites have experienced technical difficulties and data dropouts over periods ranging from few hours to several weeks. Depending on the data source, different processing steps are needed to get the data into a common format. We have to convert the point data (CPC and Snotel) to grid data, which by itself is fairly complex and time-consuming, and we must reproject grid data into a common projection, subset the dataset from its original spatial extent and populate the input grid used by the model. The data are then run through the TOPS model, which generates desired outputs.

What follows is a new step in many Earth science systems: the data are compared against long-term records and statistics, and the system determines whether there is something important happening in the covered area. An example of such events may include new fires being ignited, or rapid ice-melt and thus flooding potential. Whatever the "interesting" event is, the system tries to investigate it further, and one way of accomplishing this is by getting a higher resolution information and going through the input selection process again. The goal has now changed, not only in terms of detail, but also in geographic extent, because we no longer need to run the model over the entire continent, but only over several selected areas. Furthermore, we would like more detailed information, so we may actually choose to run a more

| Source | Variables | Frequency | Resolution | Coverage |
|---|---|---|---|---|
| Terra-MODIS | FPAR/LAI | 1 day | 1km, 500m, 250m | global |
| Aqua-MODIS | FPAR/LAI | 1 day | 1km, 500m, 250m | global |
| AVHRR | FPAR/LAI | 10 day | 1km | global |
| SeaWIFS | FPAR/LAI | 1 day | 1km x 4km | global |
| DAO | temp, precip, rad, humidity | 1 day | 1.25 deg x 1.0 deg | global |
| RUC2 | temp, precip, rad, humidity | 1 hour | 40 km | USA |
| CPC | temp, precip | 1 day | point data | USA |
| Snotel | temp, precip | 1 day | point data | USA |
| GCIP | radiation | 1 day | 1/2 deg | continental |
| NEXRAD | precipitation | 1 day | 4 km | USA |

Table 1: TOPS input data choices

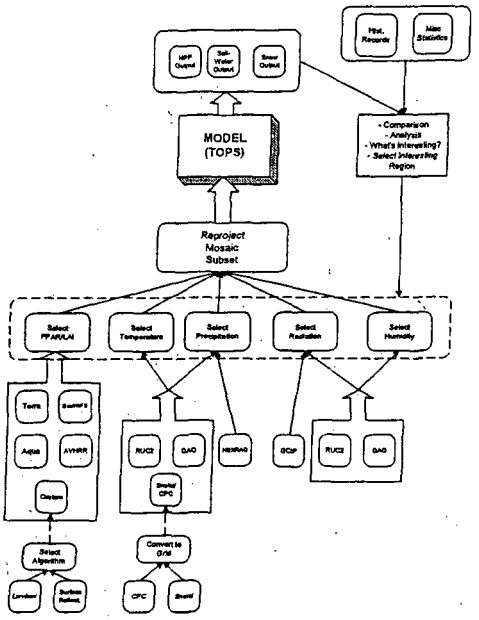


Figure 1: Data flow in the TOPS framework

complex model that runs longer, but provides us with higher quality information on the ongoing events, together with the prognosis for near future. As we can see, when this feedback loop is added to TOPS, the complexity of the system goes up even further. TOPS provides only a simple illustration of the potential problems, and is less complex than many other models and systems in the Earth sciences, some of which take dozens of different inputs, with sizes reaching into terabytes for each model run.

### 1.2 Overview

The architecture of the agent is described in Figure 2. In the remainder of the paper, we describe a few of the components of this architecture in more detail:

**DPADL** Section 2 discusses the Data Processing Action Description Language (DPADL) (Golden 2002), which is used to provide action descriptions of available programs and API calls, as well as descriptions of available data sources. DPADL is an expressive, declarative language with Java-like syntax, which allows for arbitrary constraints and embedded Java code. Planning problems are also described in DPADL.

**Planner** Section 3 discusses the planner, which accepts goals in the form of data descriptions and synthesizes data-flow programs using the action descriptions read in by the DPADL parser, consistent with information stored in the database (i.e., the initial state). It reduces the planning problem to a constraint satisfaction problem whose solution provides a solution to the original planning problem.

**Constraint Network** Section 4 discusses the constraint solver, which can handle numeric and symbolic constraints, as well as constraints over strings and even arbitrary Java objects. The latter are evaluated by executing the code embedded in constraint definitions, specified in the DPADL input file. Additionally, it can solve a limited class of universally quantified constraints (Golden & Frank 2002).

**JDAF** TOPS provides a collection of data-processing programs and scientific models for ecosystem forecasting, supporting a common API, in a framework called JDAF. There are two ways that the agent interacts with TOPS: the execution of plans and the evaluation of constraints. Certain constraints, specified procedurally using the DPADL language, are evaluated by making remote method calls to TOPS. This provides a fined-grained integration between the planner and TOPS, which is needed for the planner to compute the appropriate parameter values for TOPS API calls.

## 2 DPADL Language

In the course of developing IMAGEbot, we found that existing action representation languages were inadequate for describing data processing domains. To address these deficiencies, we developed a new language called DPADL, for Data

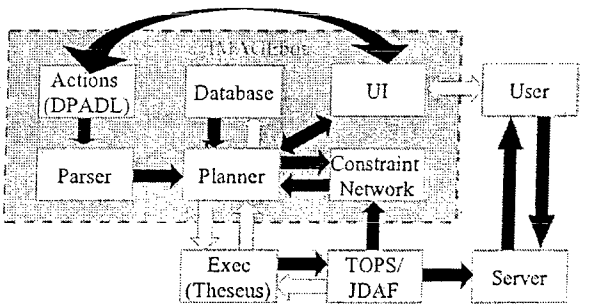


Figure 2: The agent architecture

Processing Action Description Language. DPADL provides features tailored for data processing domains, such as:

- First-class objects: Most things in the world and in software environments can be viewed as objects with certain attributes and relations to other objects. For example, a file has a name, host, parent directory, owner, etc. Even more importantly, data files often have complex data structures. The language should provide the vocabulary for describing these structures. DPADL is an object-oriented language, with a syntax based on Java and C++.
- Functions: We have found that the vast majority of predicates in data-processing domains are functions. The language should allow functions to be described as functions, rather than shoehorning everything into a relational representation. DPADL uses a functional representation. Relations are represented as functions returning boolean.
- Constraints: Determining the appropriate parameters for an action can be challenging. Parameter values can depend on other actions or objects in the plan. The language should provide the ability to specify such constraints where they are needed. DPADL supports built-in and user-defined constraints over any type, including strings and Java objects, and universally quantified constraints over sets.
- Integration with a run-time environment: It is not sufficient to generate plans; it is necessary to execute them, so there must be a way to describe how to execute the operations provided by the environment and obtain information from the environment. The language should allow the specification of "hooks" into the runtime environment, both to obtain information and to initiate operations. DPADL provides these hooks by permitting embedded Java code for defining new constraints and specifying how to execute actions. Variables used in planning and constraint reasoning can reference Java objects as well as primitives such as integers and strings, so fine-grained interaction with the Java runtime environment is possible.
- Metadata: The inputs and outputs in a data-processing domain are data files, which contain information about the world, usually at some time in the past. For example, pixels in a satellite image correspond to physical measurements of regions of the Earth at whatever time the image was captured. The language should be expressive enough to describe how the contents of existing or requested data relate to the past state of the world. In DPADL, a data product is described as a mapping from the state of the world to the contents of the data.
- Object creation and copying: Many programs create new objects, such as files, sometimes by copying or modifying other objects. The language must provide a way of describing such operations. DPADL allows effects to create new objects, which optionally may be declared as copies of existing objects, in which case it is only necessary to list the ways in which the objects differ; all other attributes are inherited from the preexisting object.
- Operations on large or infinite sets: Many programs act on all members of some set. For example, a backup operation acts on all files on a disk and an image processing operation may affect all pixels in an image, in a specified region of an image, or matching a specified criterion. The language should support universal quantification to describe such operations. DPADL provides universally quantified goals and effects, even when the sets quantified over are infinite.

## 3 Planning in the Large

Data processing has traditionally been automated by writing shell scripts. There are some situations when scripts are the best approach: namely, when the same procedure is to be applied repeatedly on different inputs, the environment is fairly stable and there are few choices to be made. However, in many applications, including TOPS, none of these assumptions holds. There are many different data products we would like the system to produce, there are many inputs and data-processing operations to choose from in producing those products, and the availability of these inputs can change over time. Additionally, the domain lends itself to planner-based automation; it has precisely characterized inputs and outputs and operations whose effects can also be precisely characterized. However, there are significant differences between Earth Science data processing and more traditional planning domains, which calls for different techniques. Notable features of data processing domains include large dynamic universes, large plans, incomplete information and uncertainty.

### 3.1 Decisions, decisions

As we discussed in Section 1.1, we have a number of inputs to choose from, which are applicable under different circumstances. The data may come from several satellites, ground stations, or as outputs from other models, forecasts and simulations.

In addition to input choices, we also have several choices of models to use with the data. As with the data, the models produce results of various quality, resolution, and geographic extent. Moreover, there may sometimes be significant trade-offs in performance versus precision. An FPAR/LAI algorithm provides a good example of this trade-off. We can produce an FPAR/LAI pixel using either a lookup table, or a radiative transfer method (Knyazikhin *et*

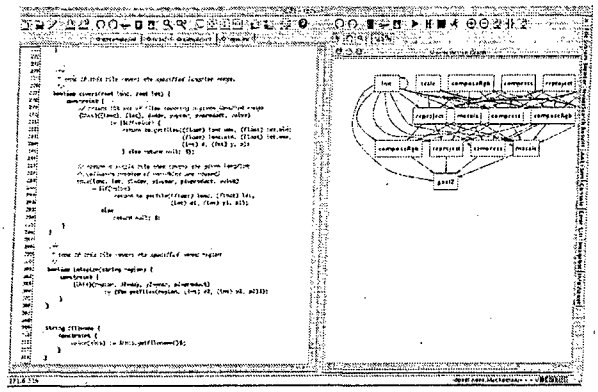

Figure 3: The IMAGEbot development environment, running as a jEdit plugin.

*al.* 1999). In the case of a lookup table, we derive a Normalized Difference Vegetation Index (NDVI) from two surface reflectance channels by a means of a simple equation, and than use the NDVI value together with its landcover value as a key into a static lookup table that will give us the FPAR and LAI values. The complexity of this algorithm is O(1). On the other hand, we can use the radiative transfer method, which contains a large number of intermediate computations and has complexity $O(n \log n)$. This fact, together with the number of runs we may attempt, translates into a substantial difference in user time, and while the radiative transfer method provides us with good results, it is not suitable for more interactive or first-pass applications, where the lookup table is sufficient. In these first-pass applications, we are looking for large abnormalities and deviations from long term normals, so high precision runs do not necessarily provide us with better results.

Another reason for using different models at different times is their possible regional character. Some models are highly specialized and provide very good and precise results in only certain parts of the world. This is partially due to the fact that the scientists who develop these models have a great deal of knowledge about specific geographic area (Pacific Northwest, the Amazons, etc.). They have collected large amounts of local data over the years, and were able to develop models whose outputs are highly accurate in these regions. We usually don't want to use these models when we are concerned with global monitoring, but they are useful when we have identified an important event occurring at the region where we have a very accurate regional model.

### 3.2 Large dynamic universes

In less than ten years, the tide in the planning community has shifted from lifted action representations to ground representations, thanks largely to the success of planners like Graphplan (Blum & Furst 1997) and HSP (Bonet & Geffner 2001) and to the benchmark planning domains made possible by the International Planning Competition. The simple fact is that, at least for these benchmark domains, planners that use ground actions are faster. There has been recent progress (Younes & Simmons 1998) in applying some of the lessons learned from these planners to speed up planners that use lifted actions, but today the fastest planners all use ground actions.

However, there are planning problems for which it is not possible to use ground actions, for example, when not all members of the universe are known at planning time. This is trivially true in information integration domains, such as (Knoblock 1996) and (Etzioni 1996), where the job of the planner is to construct a plan to consult multiple information sources, such as databases or websites, in order to answer a query. In such domains, virtually no members of the universe may be known to the planner at the time of plan generation.

In data processing domains, too, it is impossible to identify in advance all objects in the universe. Furthermore, most actions create new objects, so the universe is not even static. Browsing through the planning problems from the Third International Planning Competition (IPC3) reveals that even the hard problems typically have fewer than 100 objects total. In contrast, if we consider a single product from a single instrument (MODIS) on a single satellite (say, Terra) for a single day, there are 288 tiles. To produce a given data product, we may need to consider multiple products from multiple instruments, residing on multiple satellites, and multiple days' worth of data.

Even worse, files are not the smallest unit of granularity; they have sub-structure. For example, image-processing actions act on pixels in the image — either all pixels or a subset determined by some selection criteria. It can be very useful to describe operations at the pixel level — in fact, we do so in our own domain encodings — but doing so makes a ground representation unthinkable. A single MODIS tile contains over one million pixels. Additionally, many actions take numeric and string arguments. Appropriate values for these arguments may be determined through constraint reasoning, but there is no way to list all possible values *a priori*.

Although we cannot use a grounded representation, we would still like to benefit from some of the techniques that have been developed over the past ten years. As we discuss in Section 3.5, we adopt a lifted variant of a relaxed plan-graph analysis, combined with a constraint-based search.

### 3.3 Large plans

The purpose of data reduction is to convert large amounts of data into small amounts of information; consequently, a typical data-flow plan has a large number of inputs and a small number of outputs. Data are aggregated spatially (mosaics) and temporally (mean, max, trend analysis, animations, etc), and different data sources are fused. The plan to produce a single output may contain hundreds or thousands of actions.

While plans can grow very large, complexity need not grow accordingly. Whereas traditional benchmark problems involve a lot of interactions, making the difficulty of planning exponential in the size of the plans produced, data-processing domains are "embarrassingly parallel." Except for competition for resources such as memory and CPU, the processing required for one mosaic tile does not interfere with the processing for another tile. Indeed, even operations

on individual pixels tend to be independent of operations on adjacent pixels. This parallelism is manifest in the structure of the data-flow plans, which tend to be shallow but bushy, with many instances of the same actions operating on different inputs. Even though actions do not directly interfere with each other, there may be constraints between parameter values that arise when planning with a lifted representation. However, the CSPs corresponding to these parameter dependencies tend to be tree-structured, meaning they can be solved with no backtracking (Freuder 1982). Thus, it should be possible to generate "embarrassingly parallel" plans in time that is roughly linear in the size of the plan.

In fact, we are aiming at planning times that are sub-linear in plan size in some cases, by generating plans with simple loops that iterate over, say, all tiles matching a given set of criteria. To facilitate the detection of independence among actions and subgoals, we label certain types as "static," meaning they can be created but never modified. Detecting independence among actions that produce only static objects is trivial — unless one directly or indirectly supports the other, they are independent.

### 3.4 Incomplete information and uncertainty

There has been considerable work in planning under incomplete information and uncertainty. However, it is worthwhile to compare and contrast data processing domains with other domains that involve incomplete information.

In classical planning domains that involve uncertainty and sensing, such as the infamous bomb-in-the-toilet domain, all possible worlds are explicitly enumerated, which facilitates the case analysis necessary to solve these problems. Enumerating all possible worlds is infeasible in data processing domains, or any software domains for that matter. As we discussed in Section 3.2, one world is really too large to explicitly represent; all possible worlds is out of the question. In fact, the number of possible worlds is infinite. We adopt the Local Closed-World (LCW) reasoning introduced in (Etzioni, Golden, & Weld 1997) to efficiently reason about incomplete information in the face of very large universes. In IMAGEbot, we deal with three different kinds of uncertainty, and each is handled differently:

- Unknown information that must be known by the agent in order to complete the plan: For example, the information may be used to provide the value of a variable, or select among alternative actions. This information is sensed, not through explicit sensing actions but through the evaluation of constraints, which in turn causes code to be executed to obtain the correct values. For example, if we want to know the mosaic tiles providing a given measurement for a particular region, we can evaluate the constraint associated with the relation *tile*.covers(*lon*, *lat*) for specified intervals of *lon* and *lat* or *tile*.inRegion(*region*) for a specified named region. That, in turn, causes the Java method getTiles to be called, which connects to the TOPS sever to obtain the appropriate set of tiles. This approach cannot handle sensing actions with preconditions, because the constraints are always applicable, limited only by knowledge of the relevant variable domains. On the other hand, it affords great versatility in the manner in which information is gathered.
- Unknown information that need not be known by the agent in order to complete the plan: For example, if the user requests a file that contains gridded evening temperature values for Montana at 8 km resolution, and the agent has gridded temperatures for the western US at 1 km resolution, it need only select the appropriate subset of the data and reduce the resolution. Even though the agent never knows what the actual temperature values are, it can be confident that the file it returns to the user contains the requested information. In this sense, it is analogous to conformant planning, i.e., producing a plan that is guaranteed to work in any possible state of the world, without knowing the actual state. In fact, the metadata reasoning that the planner employs is similar to the case analysis employed by conformant/contingent planners. In order to represent that a data file contains specific information, such as temperatures, we rely on metadata formulas (Golden 2000), first-order descriptions of information sources that describe data contents in terms of the information about the world contained in the data.
- Uncertainty in how well the values stored in the data files represent the variables they are supposed to represent: Although it is tempting to represent these uncertainties in terms of probability distributions, the probabilities are unknown, even to the scientists who are experts in the field. Instead, we represent these uncertainties in an ad hoc manner, in terms of "data quality." *A priori* quality values can be assigned to data from different sources, modified by information known about specific data files. For example, satellite data have quality assurance flags, reporting problems such as cloud cover, "dead detectors," and values that are outside the expected bounds. Additionally, various processing operations can affect data quality, which we can express in terms of a mathematical relationship between the quality of the input and the quality of the output.

### 3.5 Planning approach

Space limits preclude a detailed description of the planning algorithm, but it is roughly a two-stage process. The first stage consists of a Graphplan-style reachability analysis, (Blum & Furst 1997) used to derive heuristic distance estimates for the second stage. The second stage is a constraint-based search, which is discussed in Section 4. The primary differences from Graphplan are:

- Action nodes in the graph are lifted, and each node may represent a *set* of actions of a given type. For example, in a given layer of the graph, there might be a single node corresponding to the action schema "compress(*file*)," for a thousand different instances of *file*. Nodes may be split when doing so would improve the reachability analysis, and they may, in some cases, be grounded, but in general there is not a one-to-one correspondence between nodes in the graph and actions in the final plan.
- There is no explicit proposition layer. Arcs go directly from nodes to nodes and are labeled with either individ-

ual conditions or with input-output bindings, meaning the input of the consumer is provided by the output of the producer. An input-output arc supports all preconditions of the consumer that depend on the specified input.

- The initial graph construction is backward from the goal, to avoid adding irrelevant actions. Afterward, variable bindings are propagated forward from the initial state, and unreachable nodes are eliminated.
- There are no mutexes. Computing mutexes for a lifted plan graph would be difficult and, since negative interactions are rare in data-processing domains, of little value.

After the graph is constructed, heuristic distance estimates for guiding the search are computed, and a constraint network representing the search space is incrementally built. Since the nodes in the graph represent multiple actions, these need to be copied (lazily), to avoid forcing multiple conditions to be supported by the same action. Note that, in duplicating the nodes, we are not forcing the conditions to be supported by *different* actions. We take a least-commitment approach to whether two action variables in the plan designate the same or different action. If two action variables must codesignate (or non-codesignate), this will be discovered by constraint reasoning. The constraint network contains:

1. Boolean variables for all arcs, nodes and conditions. A "true" value for an arc or a node means that element is part of the plan. A "true" value for a condition means the condition is true.
2. Variables for all parameters, input and output variables and function values.
3. For every condition in the graph, a constraint specifying when that condition holds. For conjunctive and disjunctive expressions, the constraint is the respective conjunction or disjunction of the boolean variables corresponding to appropriate sub-expressions. For equalities, inequalities, and all user-specified constraints, the constraint is the corresponding equality, inequality, etc. For fluents, no constraint is given, since the truth values of fluents are determined by the arcs that support them.
4. For every arc in the graph, constraints specifying the conditions under which the supported fluents will be achieved. These constraints consist of equality constraints between variables in the producer and consumer and preconditions that must be true before the producer action is executed.

## 4 Constraint reasoning

Constraints appear at all levels in data-processing domains.

- At the problem level, we have constraints on time and resource consumption. For example, one of the goals of the TOPS system is to perform the complete processing and analysis of data for a particular day no later than 8am the following day. If we have an algorithm that runs for 10 hours and we know that the last data for the current day will be arriving around midnight, we cannot accomplish the goal and we should consider another algorithm.
- At the file level, we can have constraints on size, quality, etc. For example, we may not want to process files that cover regions with clouds over more than 80% of the area. In this case, we may have to use a different, and less cloudy, source of data.
- At the pixel level, constraints may specify subsets of one or more datasets. For example, we may want to process data only for a certain country or region, or we may want to run an algorithm only during certain time periods. We may want to run the algorithm only on pixels of certain underlying type. For example, only for broad-leaf forests. Finally, during validation, we often compare satellite data with ground measurements, and we are only interested in specific points on the ground where we have validation measurements.

In order to deal with the many constraints that arise in a plan, we reformulate the planning problem as constraint satisfaction problem (CSP), an approach that has been investigated by researchers in an attempt to use advanced constraint solving algorithms to find plans more efficiently. In our system, we use constraint reasoning primarily for its expressive power rather than its efficiency. After the planner constructs a constraint network corresponding to the desired search space, we search the constraint network for a solution, which corresponds to a solution to the original planning problem.

A *constraint network* is a representation and reasoning framework consisting of a finite set of variables, a corresponding set of domains containing the values the variables may take, and a set of constraints. Each constraint is defined on a subset of the variables and limits the values those variables in can take simultaneously. An assignment of values to all variables that does not violate any constraints is a *solution*. The central reasoning task (or the task of solving a CSP) is to find one or more solutions.

Many algorithms and systems have been developed for solving constraint problems, ranging from simple backtracking search algorithms to sophisticated hybrid methods. However, constraint networks with infinite domains represent new challenges. In terms of representation, constraints can no longer be represented extensionally as relational tables. It is impossible to store in a computer a relation with infinite entries. From a reasoning point of view, the conventional search algorithms and consistency techniques cannot be applied directly. There is no way to enumerate values in an infinite domain exhaustively. It is unknown to us whether there is a general framework available to represent and to solve infinite constraints problems.

As discussed in Section 3, planner variables, even universally quantified variables, can have infinite domains. Since these variables can appear in constraints, we have implemented a constraint network component capable of solving a class of constraint problems with infinite domains, that is, universally quantified constraints obtained from subgoals of the planner(Golden & Frank 2002). Each variable is associated with a domain. A variable domain can be finite or infinite, in which case it is represented as an interval (for numeric type variables), a regular expression (for string type),

or symbolic sets (for object type). The use of regular expressions to represent string domains, and the support for universally quantified constraints are both novel, if somewhat unorthodox, contributions to constraint reasoning.

## 5 Java Distributed Application Framework (JDAF)

In order to facilitate interoperation of the planner with the Earth science processing algorithms, as well as general extensibility and flexibility of the overall system, we are implementing the Java Distributed Application Framework (JDAF). Using this framework we are able to easily integrate existing algorithms written in several different languages (C, C++, Fortran) into a complex application. While the algorithm integration is an important feature of the system, there is a provision for another integration, equally important — integration of the acquired data needed for the processing. There has been an enormous increase in the data volume and the number of data sources over the past several years, and while some data are being duplicated (for example we can obtain FPAR/LAI data from MODIS-Terra, MODIS-Aqua, AVHRR, or MISR), they usually come in variety of formats ranging from simple binary to HDF-EOS. The different data formats often bring another complexity into the system integration process, because the system will require new I/O modules that can read these new formats. With these facts in mind we are building our framework in a way that accommodates both data and algorithm fusion, so that we can add new algorithms and new data streams seamlessly to the existing system while minimizing the integration efforts.

Since most of the Earth science algorithms are written in C or C++, we take advantage of Java Native Interface (JNI) facilities provided by the standard Java distribution. There is a single point of entry in and out of the native code, and we only use the Java interface for parameter passing between the processing algorithm and the rest of the system. This leads to a very simple design and a fast and efficient integration. On the Java side of the system, we provide a set of common API's, which are implemented by each of the active objects (data pre-processing objects, processing algorithms, data analyzers). This makes it simple to form processing pipelines in a flexible manner, by either an application programmer, or by the planner. The simplicity of integration, flexibility, and fast deployment, makes JDAF a good candidate for prototyping of new algorithm processing systems, competing with scripting languages like Perl. Even though scripts are very suitable for fast prototypes, JDAF adds the flexibility and the distributed execution component not often available in common scripting languages.

In order to take advantage of new available data streams, we use Earth Science Markup Language (ESML), an XML-based description language that significantly eases the data fusion process, and supports one of our design goals — separating of the data from the processing algorithms. The algorithms obtain their inputs through a data translator that has a detailed knowledge about the structure of the data, while the processing component does not have to know anything about the source or the format of the data. The knowledge of the data translator comes from external XML descriptions of the data, and this external description is often the only thing needed to integrate a new data stream — no code change is required.

## 6 Conclusions

### 6.1 Related Work

There has been little work in planner-based automation of data processing. Two notable exceptions are Collage (Lansky & Philpot 1993) and MVP (Chien *et al.* 1997). Both of these planners were designed to provide assistance with data analysis tasks, in which a human was in the loop, directing the planner. In contrast, the data processing in TOPS must be entirely automated; there is simply too much data for human interaction to be practical.

Planning for data-processing shares many characteristics with planning for information integration and planner-based software agents (Golden 1998). The primary difference is the need in data-processing plans to reason about information that will never be known to the agent but is nonetheless essential to the task at hand — namely, the information contained in the data files that the agent must process.

The EnVironmEnt for On-Board Processing (EVE) (Tanner *et al.* 2001) is an execution framework for data-processing plans to be run on-board an Earth-orbiting satellite. Unlike IMAGEbot, EVE provides no planning capabilities; plans are generated by humans.

The Amphion system (Stickel *et al.* 1994) was designed to construct programs consisting of calls to elements of a software library. Amphion is supported by a first-order theorem prover. The task of assembling a sequence of image processing commands is similar to the task Amphion was designed to solve. However, the underlying representation we use is a subset of first-order logic, enabling the use of less powerful reasoning systems. The planning problem we address is considerably easier than general program synthesis in that action descriptions are not expressive enough to describe arbitrary program elements, and the plans themselves do not contain loops or conditionals.

### 6.2 Ongoing and Future Work

**Multi-criteria optimization** As we have discussed, there are many decisions to be made, in the inputs to select and the processing operations to apply to those inputs. Some decisions may be forced by parameters of the goal. If the goal calls for global historical data, regional datasets can be eliminated from consideration, as can datasets that have only been available recently. However, that can still leave a number of choices that all nominally satisfy the goal. That does not mean, however, that all of these choices are equivalent. Picking one data source over another can result in significant differences in the data product, along a number of dimensions. Which data source should be preferred depends on the users' preferences with respect to these dimensions, making the best choice a multi-criteria optimization problem. Dimensions include:

**Quality** The data can contain both spatial and temporal variability in terms of quality. This can for example hap-

pen when we have data from two different sources that were taken at slightly different times, with negative effects on the data (for example clouds) diminished from one observation to the next.

**Timeliness** Not all data sources are available in a timely fashion, so we often have a choice between poor quality data now or better quality data later. If the purpose is monitoring long-term trends, getting up-to-the-minute data is not important, but if we are interested in urgent real-time events, such as storms, we must make do with whatever datasets are currently available.

**Resolution** There are often multiple spatial and temporal resolutions to choose from. In some cases, for example global climate models, high resolution is not necessary, whereas when the focus is on smaller features, such as fires or crops, high resolution is essential.

**Resources** High-resolution data sets require more storage and bandwidth, and more processing time. Likewise, higher-accuracy models are more CPU-intensive. For example we can produce FPAR and LAI pixels using either a lookup table or a radiative transfer method (Knyazikhin *et al.* 1999). In the case of a lookup table, we derive a Normalized Difference Vegetation Index (NDVI) from two surface reflectance channels by a means of a simple equation, and than use the NDVI value together with its landcover value as a key into a static lookup table that will give us the FPAR and LAI values. The complexity of this algorithm is O(1). On the other hand, we can use the radiative transfer method, which contains a large number of intermediate computations and has complexity $O(n \log n)$. Which is preferred will depend on whether time or precision is more important.

*Multi-criteria optimization is notoriously difficult, since improving* the plan along one dimension typically degrades it along another. It is unlikely that any single plan will optimize all criteria simultaneously. For example, if any single dataset were strictly dominated by any other according to all criteria that we care about, we would just exclude that dataset from consideration from the outset. Instead, we are likely to have many candidate plans that are *Pareto optimal*, that is, plans that are impossible to improve upon in one dimension without making them worse in another. Without additional information from the user, we have no reason to prefer one Pareto optimal plan over another, so we require the user to provide a single optimization metric, such as a weighted sum of the individual criteria.

To date, we have done little to explore optimization. The latest version of DPADL allows any numeric expression to be used to specify an optimization function. This may be too expressive, since only exhaustive search of all possible plans can guarantee that a given plan is optimal. With experience of what kinds of optimization functions are used in practice, we will probably restrict this in the future.

## References


Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *J. Artificial Intelligence* 90(1–2):281–300.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Aritificial Intelligence* 129(1–2):5–33.

Chien, S.; Fisher, F.; Lo, E.; Mortensen, H.; and Greeley, R. 1997. Using artificial intelligence planning to automate science data analysis for large image database. In *Proc. 1997 Conference on Knowledge Discovery and Data Mining*.

Etzioni, O.; Golden, K.; and Weld, D. 1997. Sound and efficient closed-world reasoning for planning. *J. Artificial Intelligence* 89(1–2):113–148.

Etzioni, O. 1996. Moving up the information food chain: softbots as information carnivores. In *Proc. 13th Nat. Conf. AI.*

Freuder, E. 1982. A sufficient condition for backtrack-free search. *Journal of the Association for Computing Machinery* 29(1):24–32.

Golden, K., and Frank, J. 2002. Universal quantification in a constraint-based planner. In *Proc. 6th Intl. Conf. AI Planning Systems*.

Golden, K. 1998. Leap before you look: Information gathering in the PUCCINI planner. In *Proc. 4th Intl. Conf. AI Planning Systems*.

Golden, K. 2000. Acting on information: a plan language for manipulating data. In *Proceedings of the 2nd NASA Intl. Planning and Scheduling workshop*, 28–33. Published as NASA Conference Proceedings NASA/CP-2000-209590.

Golden, K. 2002. DPADL: An action language for data processing domains. In *Proceedings of the 3rd NASA Intl. Planning and Scheduling workshop*, 28–33. to appear.

Knoblock, C. 1996. Building a planner for information gathering: A report from the trenches. In *Proc. 3rd Intl. Conf. AI Planning Systems*.

Knyazikhin, Y.; Glassy, J.; Privette, J. L.; Tian, Y.; Lotsch, A.; Zhang, Y.; Wang, Y.; Morisette, J. T.; Votava, P.; Myneni, R. B.; Nemani, R. R.; and Running, S. W. 1999. MODIS Leaf Area Index (LAI) And Fraction Of Photosynthetically Active Radiation Absorbed By Vegetation (FPAR) Product (MOD15) Algorithm Theoretical Basis Document. http://eospso.gsfc.nasa.gov/atbd/modistables.html.

Lansky, A. L., and Philpot, A. G. 1993. AI-based planning for data analysis tasks. In *Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications (CAIA-93)*.

Nemani, R.; White, M.; Votava, P.; Glassy, J.; Roads, J.; and Running, S. 2000. Biospheric forecast system for natural resource management. In *Procceedings of GIS/EM -4*.

Nemani, R.; Votava, P.; Roads, J.; White, M.; Thornton, P.; and Coughlan, J. 2002. Terrestrial observation and predition system: Integration of satellite and surface weather observations with ecosystem models. In *Proceedings of the 2002 International Geoscience and Remote Sensing Symposium (IGARSS)*.

Stickel, M.; Waldinger, R.; Lowry, M.; Pressburger, T.; and Underwood, I. 1994. Deductive composition of astronomical software from subroutine libraries. In *Proceedings of the 12th Conference on Automated Deduction.*

Tanner, S.; Keiser, K.; Conover, H.; Hardin, D.; Graves, S.; Regner, K.; Wohlman, R.; Ramachandran, R.; and Smith, M. 2001. EVE: An on-orbit data mining testbed. In *Proceedings of the IJCAI workshop on Knowledge Discovery from Distributed, Heterogeneous, Autonomous, Dynamic Data and Knowledge Sources*.

Younes, H., and Simmons, R. 1998. On the role of ground actions in refinement planning. In *Proc. 6th Intl. Conf. AI Planning Systems*, 54–61.